Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2005-23

A TABU SEARCH HEURISTIC FOR THE QUAY CRANE SCHEDULING PROBLEM

by

Marcello Sammarra
Jean-François Cordeau
Gilbert Laporte
M. Flavia Monaco

July 2005

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone : (514) 343-7575 / Télécopieur : (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

105-06992/

PUBLICATION
CRT-2005-23

A TABU SEARCH HEURISTIC FOR THE QUAY CRANE SCHEDULING PROBLEM

by

Marcello Sammarra [1]
Jean-François Cordeau [2]
Gilbert Laporte [2]
M. Flavia Monaco [1]

The authors are grateful to Luigi Moccia for his helpful and accurate comments. Thanks are also due to the Natural Sciences and Engineering Research Council of Canada for its financial support.

[1] Dipartimento di Elettronica, Informatica e Sistemistica, Università della Calabria, 87036 Rende (CS), Italy (monaco, m.sammarra@deis.unical.it)

[2] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7, and Canada Research Chair in Distribution Management, HEC Montréal, 3000, Côte-Sainte-Catherine, Montréal, Canada H3T 2A7 (cordeau, gilbert@crt.umontreal.ca)

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2005

Abstract

This work deals with the Quay Crane Scheduling Problem, the problem of scheduling a fixed number of quay cranes in order to load and unload containers into and from a ship. The optimality criterion considered is the minimum completion time. Precedence relationships between different kinds of operations are taken into account. A tabu search heuristic is presented and results are compared with those provided by two exact algorithms.

Keywords: container terminal, crane scheduling, disjunctive graph, tabu search.

Résumé

Cet article traite du problème de construction d'horaires pour les grues dans un port de conteneurs. Le problème consiste à concevoir les horaires d'un certain nombre de grues de manière à réaliser les opérations de chargement et de déchargement des conteneurs en minimisant l'heure de fin des opérations. Des contraintes de préséance entre différents types d'opération sont considérées. Une heuristique de recherche avec tabous est présentée et les résultats obtenus sont comparés avec ceux obtenus par deux méthodes exactes.

Mots-clés : port de conteneurs, horaires, grues, graphe disjonctif, recherche avec tabous.

1 Introduction

A container transshipment port can be considered, for our purposes, as an open production system, with a material flow consisting of containers. In spite of the complexity of such a system, it is possible to represent a container transshipment port as two subsystems:

- the quayside and the landside which represent the *external interface*;
- the yard, where the containers to be loaded into a ship and those unloaded from a ship are stored, which represents the *storage system*.

Between these two subsystems, a third one, the *cargo handling system*, plays an interconnecting role: a dense network of internal equipments (cranes, straddle carriers and other sorts of specialized vehicles) is provided for container transporation inside the terminal.

It is instructive to provide a short description of container movements. Containers arriving by truck or by rail are moved to appropriate positions in the yard. The yard is partitioned into *stacks*; in practice it is sufficent to distinguish between areas dedicated to outgoing (or export) containers, ingoing (or import) containers, empty, and special (perishable or dangerous goods) containers. Outgoing containers are lifted by *yard cranes* and moved by *straddle carriers* to the ship, where they will be loaded by the *quay cranes* [14].

It is clear that optimally managing such a system is very complicated and calls for the solution of several planning problems. For this reason an appropriate solution methodology is to tackle each subproblem separately and try to restrict interactions with other subproblems. In this paper we are concerned with the planning of the quay cranes movements to load or unload ships.

The problem was first addressed by Daganzo [3] who provided a mixed integer formulation for the crane scheduling problem. The author considered a fixed number of cranes while assuming that the number of ships to be handled over the planning horizon can be fixed (static problem) or variable (dynamic problem). The problem was solved exactly for small size instances; for the larger cases a heuristic was used. Daganzo and Peterkofsky [4] later provided an exact method to speed up loading and unloading operations or to minimise delay costs. In both problems the authors define a *task* as the loading or unloading of all containers lying in a given area of the yard or the ship (*ship bay*). For this reason the tasks are considered to be preemptable and no precedence relationship is considered between containers stowed in the deck and the hold of a containership, or between loading and unloading operations. Kim and Park [6] have presented a mixed integer linear model, an exact algorithm and a heuristic for the quay crane scheduling problem. Contrary to the assumptions made in [3] and [4], the authors define a task as a group of homogeneous containers to be handled. For example, these could be containers with the same destination port. Thus in a given ship bay there may be more than one task to be handled which leads to the imposition of non-preemptable task constraints, precedence relationships between tasks and non-interference constraints between cranes. Moccia and al. [9] have studied the same problem as in [6]; the authors proposed modifications to

the model of Kim and Park and developed a branch-and-cut algorithm for its solution. Lim and al. in [8] have solved the task-to-crane assignment problem assuming non-interference between cranes; the solution method is based on dynamic programming.

In this paper, we solve the version the *Quay Crane Scheduling Problem* (QCSP) described by Kim and Park and Moccia and al. Our aim is to develop a tabu search procedure for this problem. The paper is organized as follows: in Section 2 we provide a mathematical model for the problem; Section 3 is devoted to the algorithm, followed by computational results in Section 4, and by conclusions in Section 5.

2 Notation and mathematical model

Our model for the QCSP is based on the model developed in [6] including the modifications reported in [9]. We are given a set of tasks $\Omega = \{1, \ldots, n\}$ and a set of cranes $Q = \{1, \ldots, q\}$. For convenience we introduce two dummy tasks 0 and T to represent, respectively, the first and the last task performed by each crane; thus the complete set of tasks is $\bar{\Omega} = \Omega \cup \{0, T\}$. We define p_i as the relative processing time of task $i \in \bar{\Omega}$; clearly $p_0 = p_T = 0$. Furthermore, $l_i \in \mathbb{Z}^+$ represents the location of task $i \in \Omega$ and is expressed as a ship bay number. Precedence relationships between pairs of tasks are expressed by the set $\Phi = \{(i,j) | i, j \in \Omega,\ i \text{ must be completed before } j \text{ starts}\}$, while Ψ is the set of task pairs that cannot be performed at the same time. This means that if $(i,j) \in \Psi$, then either i has to precede j or j has to be completed before j starts; so, clearly, $\Phi \subset \Psi$.

Regarding the cranes, r^k is the earliest available time of the crane k, and $l_k^0, l_k^T \in \mathbb{Z}^+$ are its starting and its final position, respectively, which are expressed by a ship bay number. Denoting by $\hat{t}$ the time needed by a crane to move between two adjacent bays, the travelling time of a crane between two generic bays can be expressed as $t_{ij} = \hat{t}|l_i - l_j|$; likewise $t_{0j}^k = \hat{t}|l_k^0 - l_j|$ and $t_{iT}^k = \hat{t}|l_i - l_k^T|$ are the travelling times taken by crane k to move from its starting position to bay j and from bay i to its final position, respectively.

We also intoduce the following variables:

- x_{ij}^k $(i, j \in \bar{\Omega},\ k \in Q)$ is equal to 1 if and only if tasks i and j are performed consecutively by crane k;
- z_{ij} $(i, j \in \Omega)$ is equal to 1 if and only if task j starts after the completion time of task i;
- c_i $(i \in \Omega)$ is the completion time of task i;

- y^k $(k \in Q)$ is the completion time of crane k;
- w is the makespan.

The model proposed in [9] is the following:

$$\text{minimise } \alpha_1 w + \alpha_2 \sum_{k \in Q} y^k \tag{1}$$

$$y^k \leq w \qquad (k \in Q) \tag{2}$$

$$\sum_{j \in \Omega} x_{0j}^k = 1 \qquad (k \in Q) \tag{3}$$

$$\sum_{j \in \Omega} x_{jT}^k = 1 \qquad (k \in Q) \tag{4}$$

$$\sum_{k \in Q} \sum_{j \in \Omega} x_{ij}^k = 1 \qquad (i \in \bar{\Omega}) \tag{5}$$

$$\sum_{j \in \Omega \cup \{T\}} x_{ij}^k - \sum_{j \in \Omega \cup \{0\}} x_{ji}^k = 0 \qquad (i \in \Omega, k \in Q) \tag{6}$$

$$c_i + t_{ij} + p_j - c_j \leq M(1 - x_{ij}^k) \qquad (i, j \in \bar{\Omega}, k \in Q) \tag{7}$$

$$c_i + p_j \leq c_j \qquad ((i,j) \in \Phi) \tag{8}$$

$$c_i + p_j - c_j \leq M(1 - z_{ij}) \qquad (i, j \in \Omega) \tag{9}$$

$$c_i + p_j - c_j + \sum_{k \in Q} \sum_{u \in \Omega \cup \{0\}, l_u \neq l_i} \hat{t} x_{uj}^k \leq M(1 - z_{ij}) \qquad (i, j \in \Omega, l_i = l_j) \tag{10}$$

$$c_j - p_j - c_i \leq M z_{ij} \qquad (i, j \in \Omega) \tag{11}$$

$$c_j - p_j - c_i - \sum_{k \in Q} \sum_{u \in \Omega \cup \{0\}, l_u \neq l_i} \hat{t} x_{uj}^k \leq M z_{ij} \qquad (i, j \in \Omega, l_i = l_j) \tag{12}$$

$$z_{ij} + z_{ji} = 1 \qquad ((i,j) \in \Psi) \tag{13}$$

$$\sum_{v=1}^{k} \sum_{u \in \Omega \cup \{0\}} x_{uj}^v - \sum_{v=1}^{k} \sum_{u \in \Omega \cup \{T\}} x_{ui}^v \leq M(z_{ij} + z_{ji}) \qquad (i, j \in \Omega, l_i < l_j, k \in Q) \tag{14}$$

$$c_j + t_{jT}^k - y^k \leq M(1 - x_{jT}^k) \qquad (j \in \Omega, k \in Q) \tag{15}$$

$$r_k - c_j + t_{0j}^k + p_j \leq M(1 - x_{0j}^k) \qquad (j \in \Omega, k \in Q) \tag{16}$$

$$x_{ij}^k \in \{0,1\} \qquad (i, j \in \bar{\Omega}, \in Q) \tag{17}$$

$$z_{ij} \in \{0,1\} \qquad (i, j \in \Omega) \tag{18}$$

$$y^k,\ c_i \geq 0 \qquad (i \in \bar{\Omega},\ k \in Q). \tag{19}$$

In this model the objective function (1) is a linear combination of the makespan (see constraints (2)) and the sum of the crane completion times; in [6] as well as in [9] it is assumed that $\alpha_1 >> \alpha_2$ because the minimisation of the makespan is much more important than the minimisation of the crane completion times; therefore we can set $\alpha_2 = 0$, minimising only the makespan. Constraints (3), (4), (6) are the classical routing constraints. Constraints (5) ensure that each task is performed by one and only one crane. Constraints (7) and (8) calculate the task completion times and simultaneously eliminate subtours. The correct completion times for the last and the first task on each crane are computed through constraints (15) and (16). Constraints (9), (10), (11) and (12) define the z_{ij} variables and prevent crane interference, which occurs whenever the completion time of a task i is equal to the starting handling time of another task j, the tasks lie in the same bay, and are performed by different cranes. In this case the crane must perform task i safely, which can be enforced by starting task j no sooner than $c_i + \hat{t}$. Constraints (13) state that tasks i and j cannot be handled at the same time if $(i, j) \in \Psi$. Finally constraints (14) avoid interferences between cranes due to overtaking. The constant M in the model is a large number.

3 Algorithm

The QCSP can formally be defined as a scheduling problem on *parallel uniform machines* with precedence constraints, denoted in the classic three fields notation as $Pm|prec|C_{max}$. The QCSP also possesses non-standard characteristics: ready times on machines and explicit non-simultaneity constraints. It is well known [11] that $Pm|prec|C_{max}$ is strongly NP-hard unless the precedence constraints graph is an *outtree* or an *intree*; other polynomially solvable cases of the above problem arise when the number of machines is one or greater than the number of jobs. Thus the *QCSP* is at least as difficult as $Pm|prec|C_{max}$ since it reduces to this problem when $r^k = 0 \; \forall k \in K$ and $\Psi = \Phi$. The QCSP can also be viewed as a *vehicle routing and scheduling problem* consisting of two subproblems:

- a *routing* problem, which determines the *sequence* of tasks on each machine;
- a *scheduling* problem, which determines, for each *sequence*, the starting handling time (or equivalently the completion time) of the tasks belonging to that route.

In [6] a branch-and-bound algorithm and a *Generalized Random Adaptive Search Procedure* (GRASP) are used to solve the QCSP, while in [9] the authors present an exact branch-and-cut

algorithm. The branch-and-bound method of Kim et al. as well as the GRASP heuristic work very well on instances with two cranes and ten to 15 tasks, while they fail on instances with three cranes and 20 to 25 tasks. In contrast the branch-and-cut algorithm of Moccia et al. has a higher overall efficiency: it outperforms the branch-and-bound algorithm, both in terms of solution quality and of computation time. Clearly the fast growth of the solution time with the size of the instances remains the major drawback of the branch-and-cut algorithm, although it is faster than the branch-and-bound algorithm. The heuristic algorithm presented in this paper offers, in contrast, a reasonable compromise in terms of computational burden and solution quality. It results from tackling the QCSP as a vehicle routing problem and it is based on tabu search [5].

In what follows, we refer to the task sequence handled by a crane k as a *route* σ^k and we denote a schedule by $\sigma = (\sigma^1, \ldots, \sigma^q)$. Moreover given a route σ^k we denote by σ^{k-1} and σ^{k+1}, respectively, the left adjacent and the right adjacent route of $\sigma^k \quad \forall k \in Q \setminus \{1, q\}$. The first route has no left adjacent route and the last route has no right adjacent route. The generic task performed by crane k (a vertex in the route σ^k) is denoted by σ_i^k; a path from 0 to T is denoted by $\pi = (\pi_0, \ldots, \pi_T)$. The scheduling and routing subproblems are addressed in Sections 3.1 and 3.2, respectively.

3.1 A local search algorithm for the scheduling subproblem

When feasible values of the x variables are given, i.e., for a given task sequence, what remains to be done is to calculate the completion times of the tasks. The latter subproblem can be modeled on a graph $G = (\bar{\Omega}, E)$ where $E = A \cup E_\Psi$, $A = \bigcup_{k \in Q} A^k \cup A_\Phi$ with :

$$A^k = \left\{(i,j) : x_{ij}^k = 1, i, j \in \bar{\Omega}\right\} \ \forall k \in Q, \ A_\Phi = \Phi \setminus \bigcup_{k \in Q} A^k, \ E_\Psi = \Psi \setminus \bigcup_{k \in Q} A^k.$$

For example, consider the following instance:

Task	1	2	3	4	5	6	7	8	9	10
Processing time	1	32	50	13	33	26	55	4	30	7
Ship bay	2	5	6	7	8	8	9	9	10	10
Crane 1	$l_0^1 = 0$		$r^1 = 0$							
Crane 2	$l_0^2 = 6$		$r^2 = 0$							

$\Phi = \{(5,6), (7,8), (9,10)\}$

$\Psi = \{(2,3), (3,4), (4,5), (4,6), (5,6), (5,7), (5,8), (6,7), (6,8), (7,8), (7,9), (7,10), (8,9), (8,10), (9,10)\}$

The graph G corresponding to $x^1_{01} = x^2_{12} = x^1_{24} = x^1_{45} = x^1_{57} = x^1_{79} = x^1_{9T} = x^2_{03} = x^2_{36} = x^2_{68} = x^2_{8\,10} = x^2_{10\,T} = 1$ is depicted in Figure 1. In this graph:
$A^1 = \{(0,1),(1,2),(2,4),(4,5),(5,7),(7,9),(9,T)\}$, $A^2 = \{(0,3),(3,6),(6,8),(8,10),(10,T)\}$, $A_\Phi = \Phi$ and $E_\Psi = \{(2,3),(3,4),(4,6),(5,8),(6,7),(7,10),(8,9)\}$.

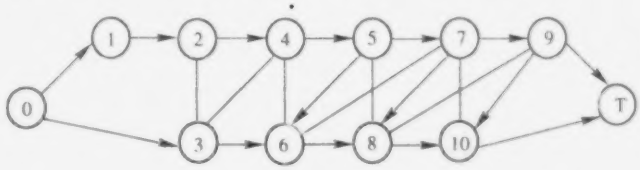

Figure 1: The disjunctive graph obtained from the values of the x variables.

Graph G consists of $|Q|$ *chains* connected to each other by arcs and edges (undirected arcs). Such a graph is known as a *disjunctive graph*.

3.1.1 Disjunctive graph representation

The concept of disjunctive graph was introduced by Roy and Soussmann [13] for the shop scheduling problem $Jm||C_{max}$. In this context two represention schemes are available. In both representations the vertices represent operations of a single task. The difference between the two representations lies in the meaning of a chain. In the first representation a chain is a task. Therefore, the vertices belonging to the same chain are connected by arcs while those belonging to different chains are connected by edges. In the second representation a chain is the set of operations performed by a machine. Therefore, vertices belonging to the same chain are connected by edges while vertices lying on different chains are connected by arcs, as the correct sequence of operations for each task must be kept. In both representations every feasible schedule for the job shop problem corresponds to an *edge orientation* yielding an acyclic directed graph; conversely any orientation yielding an acyclic direct graph corresponds to a feasible schedule [12]. Moreover the makespan of a feasible schedule corresponding to an acyclic orientation of the edges is determined by the *longest path* from a dummy source vertex 0 to a dummy sink vertex T. Thus, the minimisation of the makespan in a job shop enviroment reduces to finding a feasible orientation minimising the longest path length [11].

The same idea can be applied to the QCSP, although the structure of the underlying disjunctive graph is quite different from that of the job shop problem. Actually, while the chains

of the job shop graph are only connected by edges or by arcs, in our case both edges and arcs can exist between chains. This is not a drawback as this structure can be profitably exploited to reduce the number of edges. Consider vertices 4 and 6 connected by an edge; because there exists a direct path between the nodes 4 and 6, any feasible orientation will include arc $(4, 6)$ but not arc $(6, 4)$, for otherwise a cycle between nodes 4, 5 and 6 would result; the same reasoning can be applied to edges $(5, 8)$ and $(7, 10)$. The resulting graph is depicted in Figure 2. When

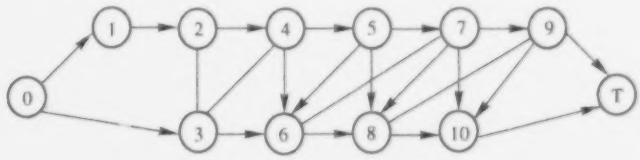

Figure 2: The reduced disjunctive graph.

a feasible orientation can be determined for the edges, as just shown, computing the makespan reduces to determining a longest path from 0 to T on a weighted directed graph, were the arc weights are given by:

$$\begin{aligned}
&\tau_{0j} = r_0^k + t_{0j}^k + p_j && \forall k \in Q,\ x_{0j}^k = 1,\\
&\tau_{iT} = t_{iT}^k && \forall k \in Q,\ x_{iT}^k = 1,\\
&\tau_{ij} = t_{ij} + p_j && \forall k \in Q,\ (i,j) \in A^k,\\
&\tau_{ij} = p_j && \forall (i,j) \in A_\Phi \cup \bar{E}_\Psi.
\end{aligned}$$

The longest path can be computed through a label setting algorithm derived from the general Bellman-Ford algorithm [1].

3.1.2 Local search heuristic

When it is not possible to direct all the edges, as in the example just provided, we use the following simple local search heuristic. Let $U = \{u_1, \ldots, u_\lambda\}$ be the set of undirected edges, $F = \{f_1, \ldots, f_\lambda\}$ a random orientation of U, and F^i $(i = 1, \ldots, \lambda)$ the orientation obtained from F by reversing the orientation of the arc f_i.

1. **Initialization**

   a) *best* $= \infty$

b) compute a longest path in the graph $G = (\bar{\Omega}, A, F)$; let *cost* be the solution value

c) if *cost* < *best* then *best* = *cost*

2. **Local Search**

a) for $i = 1, \ldots, \lambda$

- compute a longest path in the graph $G = (\bar{\Omega}, A, F^i)$
- if *cost* < *best* then *best* = *cost*

b) return *best* and the corresponding solution.

The approximation introduced by the local search procedure, which solves only $|U| + 1$ longest path problems instead of $2^{|U|}$, is very drastic. Motivations for this approach arise from the following considerations: first, it is suitable to overstimate the optimal value rather than using a lower bound on the makespan; second, the computation of a good upper bound must be carried out quickly. The ideas of the solution approach presented in this section are integrated within a tabu search heuristic to be described in Section 3.2.

3.2 A tabu search algorithm for the routing subproblem

We now propose a tabu search algorithm for the routing subproblem. The basic ingredient of a tabu search heuristic is the neighbourhood function Δ which enables moves between different feasible solutions. Our neighbourhood function transforms a feasible schedule σ into the set of schedules $\Delta(\sigma)$ obtained by performing one of the following moves:

- *swapping* adjacent tasks;
- *inserting* a task, currently assigned to route k, into an adjacent route.

Obviously the running time of any tabu search algorithm depends on the size of the neighbourhood, since the objective function must be evaluated for each neighbour. In our case the evaluation of the objective function is not immediate because it requires the solution of an NP-hard problem. We have to consider a small, but at the same time good, neighbourhood structure.

3.2.1 The swapping move

Consider a schedule $\sigma = (\sigma^1, \ldots, \sigma^q)$. If we perform all possible swaps on σ, we obtain a neighbourhood whose size is bounded from above by the number of tasks. However some swaps lead to infeasible solutions because the resulting schedule may violate a precedence constraint; other swaps may yield makespans which are worse than the current value. We now expalain the swapping move.

The value of the makespan equals the value of a longest path π in G; the longest path determines the *critical tasks*, i.e., those tasks whose starting time cannot be postponed without increasing the makespan. Tasks lying on the critical path can be grouped into *blocks* defined as the maximum number of tasks lying on the critical path and handled by the same crane. For the example reported in the Section 3.1.1, suppose that, for a given orientation, the longest path is $\pi = (0\,,1\,,2\,,3\,,4\,,5\,,6\,,7\,,9\,,T)$ (see Figure 3).

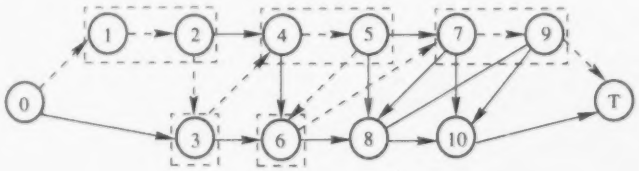

Figure 3: Graph showing blocks and the critical path identified by dashed arcs.

There are five blocks: $P^1 = [1,2]$, $P^2 = [3]$, $P^3 = [4,5]$, $P^4 = [6]$, $P^5 = [7,9]$. If tasks 8 and 10 are swapped on σ^2, the path π still remains in the graph resulting after the swap. This amounts to saying that swapping tasks not belonging to the longest path does not immediately improve the makespan. This is why Van Laahroven et al. [7] have proposed swapping only those tasks belonging to the critical path while Nowicki and al. [10] suggested swapping only the first and the last two tasks of each block, leading to an important neighbourhood reduction. Our swapping rule is quite different: for each block $P^h = \pi_{h_1}, \ldots, \pi_{h_u}, u \geq 2$, and for each pair of consecutive tasks in P^h, if $\Phi(\pi_{h_i}, \pi_{h_{i+1}}) = 0$ then swap tasks π_{h_i} and $\pi_{h_{i+1}}$. We do not allow the swapping of the last task of block P^h with the first one of block P^{h+1} because this swap corresponds to an insertion move.

3.2.2 The insertion move

While it is easy to evaluate swapping moves that do not improve the makespan, the same cannot be said of insertion moves. However, we found that the best insertions are those that

limit overcrossing between cranes. Thus we allow the movement of tasks from their currently assigned crane to an adjacent one. More precisely we evaluate the following insertion moves:

1. insertion of a task i from route σ^k as the first one in route σ^{k+1}, $k = 1, \ldots, q-1$;
2. insertion of a task i from route σ^k as the last one in route σ^{k+1}, $k = 1, \ldots, q-1$;
3. insertion of a task i from route σ^k as the first one in route σ^{k-1}, $k = 2, \ldots, q$;
4. insertion of a task i from route σ^k as the last one in route σ^{k-1}, $k = 2, \ldots, q$.

If the insertion of task i in the first position of an adjacent route is infeasible, meaning that in the route there is at least a task j such that $(j, i) \in \Phi$, we then insert task i immediately after the last task j such that $(j, i) \in \Phi$; also if the insertion move on the last position of an adiacent route is infeasible, we then insert i immediately before the first task j for which $(i, j) \in \Phi$.

3.2.3 The search process

Tabu search uses a memory process [5] designed to overcome the propensity of general local search algorithms of becoming trapped in a local optimum. To this end tabu search algorithms must be able to perform moves that deteriorate the current solution with the hope of eventually identifying a better one. To prevent the search from revisiting previously visited solutions, the algorithm keeps in memory the most recent moves, declaring those as forbidden, or *tabu*, for a given number of iterations. Forbidden moves may lose their tabu status if they yield improved solutions. To describe how our tabu search algorithm works, it is necessary to introduce additional notation. A schedule σ is represented by a set of attributes $B(\sigma) = \{(i, j, k) | i \text{ and } j \text{ are consecutive tasks handled by crane } k\}$. As in [2] the transition from a schedule σ to another schedule $\bar{\sigma}$ can be carried out by modifying $B(\sigma)$ according to the neighbourhood function described in 3.2.3. If an attribute (i, j, k) is removed from $B(\sigma)$, then (i, j, k) is declared tabu and cannot be inserted in $B(\sigma)$ before θ iterations. This means that a move is considered tabu if at least one of the attributes, describing the schedule obtained by that move, is tabu. Neverthless a tabu move will be performed if at least one attribute satisfies the *aspiration criterion*, i.e., the move yields a new incumbent. The aspiration level of an attribute (i, j, k) will be indicated by α_{ij}^k. The number of iterations for which an attribute remains forbidden is denoted by β_{ij}^k, while the number of times (*frequency*) that an attribute

has been added to a solution is denoted by δ_{ij}^k. The frequency of attributes is used to diversify the search. In order to diversify the search, non-improving moves are penalized with a term proportional to the sum of the frequencies of all the attributes added to the current solution.

In what follows the neighbourhood of a schedule σ will be denoted by $N(\sigma)$, while the set of schedules obtained by performing non-tabu moves will be denoted by $M(\sigma)$. Finally we recall that w is the makespan of a schedule σ, while w^* and σ^* represent, respectively, the incumbent optimal value of the makespan and the schedule for which the value is reached, and λ is an iteration counter.

3.2.4 Overall description of the tabu search heuristic

Our tabu search aglorithm for the QCSP can now be described:

1. Compute a starting solution σ; let w the value of the makespan.
2. $\sigma^* = \sigma$, $w^* = w$.
3. $\forall (i,j,k)\ \delta_{ij}^k = 0$, $\beta_{ij}^k = 0$, $\alpha_{ij}^k = \infty$.
4. $\forall (i,j,k) \in B(\sigma)\ \ \alpha_{ij}^k = w^*$.
5. For $\lambda = 1, \ldots, \lambda_{max}$
   a) $M(\sigma) = \emptyset$.
   b) $\forall \bar{\sigma} \in N(\sigma), \forall (i,j,k) \in B(\bar{\sigma}) \setminus B(\sigma)$ such that $\beta_{ij}^k < \lambda$ or $\bar{w} < \alpha_{ij}^k$, $M(\sigma) = M(\sigma) \cup \{\bar{\sigma}\}$.
   c) $\forall \bar{\sigma} \in M(\sigma)$, if $\bar{w} \geq w$, then $g(\bar{\sigma}) = \bar{w} + \gamma w \sum_{(i,j,k) \in B(\bar{\sigma}) \setminus B(\sigma)} \delta_{ij}^k / \lambda$, else $g(\bar{\sigma}) = \bar{w}$.
   d) Solve $\min_{\bar{\sigma} \in M(\sigma)} g(\bar{\sigma})$, and let $\hat{\sigma}$ be the optimal value.
   e) $\forall (i,j,k) \in B(\sigma) \setminus B(\hat{\sigma})$ set $\beta_{ij}^k = \lambda + \theta$.
   f) $\forall (i,j,k) \in B(\hat{\sigma}) \setminus B(\sigma)$ set $\delta_{ij}^k = \delta_{ij}^k + 1$.
   g) If $\hat{w} < w^*$, then $\sigma^* = \hat{\sigma}$ $w^* = \hat{w}$.
   h) $\forall (i,j,k) \in B(\hat{\sigma})$ set $\alpha_{ij}^k = \min\{\alpha_{ij}^k, \hat{w}\}$.
   i) Set $\sigma = \hat{\sigma}$ and $w = \hat{w}$.

4 Computational results

The tabu search algorithm described in Section 3.2 was coded in C++ and run on a PC equipped with a P4 2.66GHz processor and 256Mb of RAM. It depends on three parameters: the maximum number of iterations allowed λ_{max}, the length of the tabu list θ, and the coefficient γ of the diversification process in Step 5.c. The fine tuning of the parameters was carried out by running the algorithm with various values of λ_{max}, θ and γ. Regarding γ, one observes that this parameter should depend on the problem size. After some experimentation we set set $\gamma = \hat{\gamma}\sqrt{|\Omega||Q|}$, with $\hat{\gamma}$ equal to 0.15, and then $\lambda_{max} = 5000$. With these values, we let θ vary in the interval $[5, 15]$.

While small values of θ, around 7, are suitable for small and medium size instances, for the larger instances this setting is unsuitable. We observed that for large size instances, the number of iterations required to find the best solution decreases with θ, while in small size instances the number of iterations is not affected by θ. Thus we fixed $\theta = 12$. Finally we recall that the performance of tabu search, like the performance of any local search heuristic, is affected by the starting solution, although in tabu search this influence is limited by the diversification mechanism. We have run the algorithm with two different starting solutions. A first one, *S-TASKS*, is constructed by assigning to each crane the same number of tasks. The second starting solution, *S-LOAD*, is obtained by assigning tasks to cranes so that each crane has the same load in terms of total processing time. We found that the best results were reached by setting $\theta = 12$, $\hat{\gamma} = 0.05$ for *S-TASKS* and $\theta = 12$, $\hat{\gamma} = 0.01$ for the *S-LOAD* starting solution.

The algorithm was tested on the instances used by [6] and [9]. The dimensions of the instances are the following:

- instances from $k13$ to $k22$ have 10 tasks and 2 cranes (set A);
- instances from $k23$ to $k32$ have 15 tasks and 2 cranes (set B);
- instances from $k33$ to $k42$ have 20 tasks and 3 cranes (set C);
- instances from $k43$ to $k49$ have 25 tasks and 3 cranes (set D).

The comparison results between our algorithm, the branch-and-cut algorithm, the branch-and-bound algorithm and GRASP heuristic are summarized in Table 2, where "Best" indicates the best value reached by the algorithms considered and "Time" is the computation time required by the algorithms to find the best solution and is expressed in minutes. The branch-and-cut algorithm was run on a 2.5 GHz P4 machine equipped with 512 Mb of RAM [9];

Set	Kim and Park [6]				Moccia et al. [9]		Tabu Search			
	GRASP		Branch-and-Bound		Branch-and-Cut		S-TASKS		S-LOAD	
	Best	Time	Best	Time	Best	Time	Best	Time	Best	Time
A	488,4	0,352	488,1	0,435	459,6	2,63	459,6	1,46	459,6	1,44
B	678,6	1,462	667,5	17,525	666,6	12,219	667,8	5,84	669	5,79
C	651,9	3,162	###	646,486	604,5	72,184	610,8	22,02	610,2	21,69
D	836,5	7,564	###	808,71	801,8	102,48	803,5	50,04	798,8	52,61

Table 1: Computational comparison of three algorithms on four sets of instances: average values (### means that average values cannot be computed).

the branch-and-bound algorithm and the GRASP heuristic were run on a 466MHz PII machine equipped with 64Mb of RAM [6]. When no optimal solution is reached by the exact algorithms, for the branch-and-bound algorithm a lower bound and an upper bound are reported (instances ranging from $k35$ to $k49$); for the branch-and-cut the best solution and the gap are reported (instances $k31$, $k39$, $k41$, $k43$, $k44$, $k45$, $k48$, $k49$). Average values are reported in Table 1.

Comparing results obtained by tabu search with those reported in [6], we note that branch-and-bound yields a better solution than tabu search for only seven instances out of 39, when it is initialized with *S-LOAD*. This number decreases to five when we start with an *S-TASKS* solution. In all other cases (32 out 39 and 34 out of 39, respectively) the solution value provided by tabu search is better than or equal to that obtained in [6]. Moreover it is worth observing that for instances ranging from $k35$ to $k49$ our algorithm is much better than the branch-and-bound algorithm, both in terms of solution quality and computation time. Actually for these instances the branch-and-bound algorithm is unable to find an optimal solution, while our algorithm finds a solution of value lower than the upper bound provided by the branch-and-bound algorithm. In comparison with the GRASP heuristic, tabu search is slower but identifies better solutions. In only six instances out of 36 do the GRASP and tabu search heuristics yield the same solution value; in all other cases tabu search provides a better solution.

A close look at the results reported in the rightmost columns of Table 2 shows that our algorithm works very well, when compared with the branch-and-cut algorithm, on data sets A and D, i.e., for the smallest and the largest instances tested. Indeed in these cases tabu search is capable of finding an optimal solution on several instances. For medium size instances (data

		Kim and Park [6]				Moccia et al. [9]		Tabu Search			
								S-TASKS $\hat{\gamma} = 0.05, \theta = 12$		*S-LOAD* $\hat{\gamma} = 0.01, \theta = 12$	
Instance	Set	Branch-and-Bound		GRASP		Branch-and-Cut					
		Best	Time	Best	Time	Best	Time	Best	Time	Best	Time
k13	A	453	0,33	453	0,47	453	17,81	453	1,31	453	1,30
k14	A	546	0,28	546	0,28	546	0,03	546	1,55	546	1,51
k15	A	513	0,22	516	0,18	513	0,02	513	1,34	513	1,34
k16	A	321	1,07	321	0,25	312	0,21	312	1,47	312	1,47
k17	A	456	0,47	456	0,38	453	0,05	453	1,46	453	1,45
k18	A	375	0,58	375	0,52	375	0,03	375	1,51	375	1,50
k19	A	552	0,73	552	0,37	543	5,20	543	1,38	543	1,38
k20	A	480	0,28	480	0,37	399	0,11	399	1,53	399	1,53
k21	A	465	0,02	465	0,27	465	0,01	465	1,50	465	1,49
k22	A	720	0,37	720	0,43	537	0,18	537	1,50	537	1,50
k23	B	576	6,20	591	1,08	576	0,16	576	5,72	582	5,76
k24	B	669	6,13	675	1,57	666	0,09	672	5,83	672	5,78
k25	B	738	37,98	741	1,47	738	0,35	741	5,87	741	5,81
k26	B	639	1,30	651	2,10	639	0,15	639	5,82	645	5,71
k27	B	657	3,12	687	0,87	657	0,05	660	5,62	660	5,63
k28	B	537	66,83	549	1,38	531	0,85	531	5,92	531	5,87
k29	B	807	10,85	819	1,57	807	0,21	810	5,73	810	5,67
k30	B	891	0,20	906	1,62	891	0,07	891	5,79	891	5,80
k31	B	570	41,13	570	0,95	570 (0,53)	120,00	567	5,93	567	5,88
k32	B	591	1,52	597	2,02	591	0,26	591	6,15	591	6,00
k33	C	603	149,55	666	1,83	603	10,64	603	21,14	612	20,94
k34	C	717	147,95	762	1,97	717	14,86	741	22,29	735	22,15
k35	C	678-690	683,56	699	2,12	684	42,04	690	23,89	690	21,87
k36	C	678-720	772,58	708	3,97	678	86,07	684	21,74	681	21,50
k37	C	510-516	732,68	540	4,65	510	21,17	516	22,38	516	22,17
k38	C	609-633	645,82	660	2,67	618	120,01	618	21,79	618	21,19
k39	C	513-552	785,42	579	3,90	513 (0,9)	120,00	519	21,53	519	21,41
k40	C	558-576	579,05	597	2,47	564	67,07	567	22,38	567	22,05
k41	C	585-654	564,88	642	4,48	588 (0,5)	120,01	594	21,55	594	22,28
k42	C	561-588	587,73	666	3,57	570	120,01	576	21,54	570	21,42
k43	D	864-951	815,63	942	4,85	897 (4,2)	120,01	888	58,39	879	58,85
k44	D	813-879	660,82	858	10,32	822 (0,2)	120,01	834	49,71	834	69,83
k45	D	825-861	959,47	873	6,65	840 (1,8)	120,01	849	49,47	843	49,34
k46	D	687-708	408,43	735	11,12	690	90,37	690	48,31	690	47,32
k47	D	789-912	727,97	807	7,28	792	26,95	792	46,23	792	45,71
k48	D	627-669	932,77	669	5,12	645 (2,5)	120,01	666	49,98	642	49,39
k49	D	888-915	1162,85	972	7,62	927 (5,1)	120,01	906	48,25	912	47,88

Table 2: Computational comparison of four algorithms: detailed results

sets B and C) our method solves eight and six instances when initialized with the *S-TASKS* or with the *S-LOAD* solution, respectively. In both cases, the algorithm deteriorates on instances B. However when an optimal solution is not reached, the solution value found by tabu search is very close to the optimal value obtained in [9]. Actually the largest gap is approximately 3% (instance $k34$, *S-TASKS*). Regarding computation times it is clear that our heuristic is disavantaged with respect to the branch-and-cut algorithm because we use a stopping criterion that is not based on the proof of the optimality of the solution found. Even so, the performance of our algorithm is comparable to that of the branch-and-cut algorithm; in fact for medium and large instances, data sets C and D, respectively, in only four cases (problems $k33$, $k34$, $k37$, $k47$) is branch-and-cut faster than tabu search.

5 Conclusions

We have described a model and an algorithm for the Quay Crane Scheduling Problem. We have proved that QCSP can be viewed as a vehicle routing problem which can be decomposed into a routing problem and a scheduling problem. The scheduling problem is NP-hard, because it reduces to a longest path problem in a disjunctive graph. We have developed a tabu search algorithm for the routing problem and we have embedded into it a local search technique for the scheduling problem. Our results have been compared with those provided by a GRASP heuristic, a branch-and-bound algorithm and a branch-and-cut algorithm. We have seen that our algorithm outperforms the GRASP heuristic and the branch-and-bound algorithm. Compared with the branch-and-cut, our algorithm provides a good compromise between solution quality and computation time. Our tabu search is capable of identifying the optimal solution on several instances. When an optimum is not found, the solution obtained is almost optimal.

Acknowledgements

This work was partially done when the first author was visiting the Center for Research on Transportation, Université de Montréal. We are grateful to Luigi Moccia for his helpful and accurate comments. Thanks are also due to the Canadian Natural Sciences and Engineering Research Council for its financial support (grants 227837-04 and OGP0039682).

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows - Theory, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30:105–119, 1997.

[3] C. F. Daganzo. The crane scheduling problem. *Transportation Research - B*, 23:159–175, 1989.

[4] C. F. Daganzo and R. I. Peterkofsky. A branch and bound solution method for the crane scheduling problem. *Transportation Research - B*, 24:159–172, 1990.

[5] F. Glover and M. Laguna. Tabu search. In C.R. Reeves, editor, *Modern Heuristics Techniques for Combinatorial Problems.* Blackwell, Oxford, 1993.

[6] K.H. Kim and Y.M. Park. A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156:752–768, 2004.

[7] P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[8] A. Lim, B. Rodrigues, F. Xiao, and Y. Zhu. Crane scheduling with spatial constraints. *Naval Research Logistics*, 51:386–406, 2004.

[9] L. Moccia, J.-F. Cordeau, M. Gaudioso, and G. Laporte. A branch-and-cut algorithm for the quay crane scheduling problem. Publication 2004-04, Center for Research on Transportation, 2004.

[10] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.

[11] M. Pinedo. *Scheduling - Theory, Algorithms and Systems.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[12] M. Pinedo and M. Singer. A shifting bottelneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46:1–17, 1999.

[13] B. Roy and B. Soussmann. Les problèmes d'ordonnancement avec constraintes disjonctives. Note D.S. 9 bis, SEMA, Paris, 1964.

[14] D. Steenken, S. Voss, and R. Stahlbock. Container terminal operation and operations research - a classification and literature review. *Operations Research Spectrum*, 26:3–49, 2004.